RFM69

Generato da Doxygen 1.8.13

Indice

Capitolo 1

Driver per i moduli radio RFM69

La classe **RFM69** (pag. 13) permette di collegare due microcontrollori tramite moduli radio della famiglia RFM69 di HopeRF, e in particolare tramite il modulo RFM69HCW (`http://www.hoperf.com/rf_transceiver/modules/RFM69HCW.html`). Non ho eseguito alcun test sugli altri moduli.

Alla fine di questo testo si trova un esempio dell'utilizzo di questa classe.

Caratteristiche del modulo radio

Caratteristiche principali dei moduli radio RFM69HCW:

- frequenza: 315, 433, 868 oppure 915 MHz (esistono quattro versioni per adattarsi alle bande utilizzabili senza licenza in diversi paesi)
- potenza di emissione: da -18dBm a +20dBm (100mW)
- sensdibilità: fino a -120dBm (con bassa bitrate)
- bitrate fino a 300'000 Baud
- modulazioni: FSK, GFSK, MSK, GMSK, OOK

I messaggi possono includere un controllo CRC16 di due bytes che riduce drasticamente la probabilità di errore durante la trasmissione. Possono inoltre essere criptati secondo l'algoritmo Avanced Encryption Standard AES-128 con una chiave di 16 bytes per impedirne la lettura da parte di eventuali terze radio. La possibilità offerta dal modulo di assegnare ad ogni modulo un indirizzo unico in modo da creare una rete con fino a 255 dispositivi nonb è sfruttata, ma un risultato simile è ottenibile creando una rete in cui ogni radio ha una sync word unica che può sostituire temporaneamente con quella di un'altra radio (ottenuta da una tabella pubblica) per inviare un messaggio a quella radio. Questo permette di creare una rete di dimensione arbitraria (è possibile impostare fino a 8 byte di sync word, per un totale di 2^64 indirizzi possibili) ma non di inviare messaggi broadcast, come invece il sistema di addressing incluso nel modulo permetterebbe.

Corrente di alimentazione richiesta (a 3.3V), per modalità:

- Sleep: 0.0001 mA
- Standby: 1.25 mA
- Rx: 16 mA

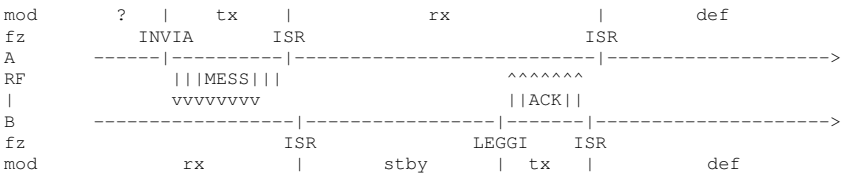
- Tx: 16 - 130 mA a seconda della potenza di trasmissione

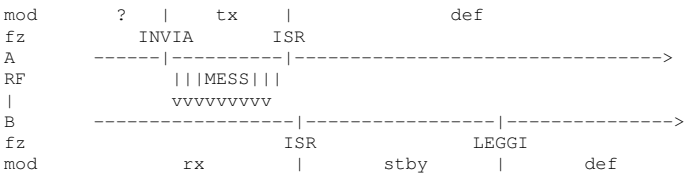
Protocollo di comunicazione

Il protocollo di comunicazione alla base di questa classe presuppone che in una stessa banda di frequenza siano presenti esattamente due radio che condividono la stessa sync word. La stessa frequenza può quindi essere utilizzata anche da altri dispositivi; naturalmente, però, se dispositivi trasmittenti sulla stessa frequenza trasmettono dati nell stesso momento nessuno di essi riceverà un mesaggio valido (a meno che la differenza nella potenza trasmessa sia abbastanza grande da permettere al segnale più forte di "coprire" il più debole, in tal caso solo il dispositivo ricevente il più forte otterrà un messaggio).

Alla lettura di ogni messaggio la radio ricevente può trasmettere automaticamente un segnale di ACK se la radio trasmittente lo ha richiesto. In questo modo se l'utente deve essere certo che un messaggio trasmesso sia stato ricevuto e letto (quindi certamente anche utilizzato, visto che la lettura avviene solo su richiesta dell'utente e non automaticamente come la ricezione) non deve né implementare un sistema di ACK né modificare il codice ricevente, e il segnale di ACK sarà il meno dispendioso possibile in termini di tempo del programma.

Gli schemi sottostanti illustrano la trasmissione di un mesasggio. Nel primo caso si tratta di un messaaggio con richiesta di ACK, nel secondo no.





- `A, B`: Programma delle stazioni radio, evoluzione nel tempo
- `fz`: funzioni chiamate. `invia()` e `leggi()` sono chiamate dall'utente, `isr()` è l'interrupt service routine della classe
- `mod`: modalità della radio. `tx` = trasmissione, `rx` = ricezione, `stby` = standby, `def`: la modalità che l'utente ha scelto come default per quella radio
- `RF`: presenza di segnali radio e loro direzione

Collisioni

Le funzioni di questa classe non impediscono che le due radio trasmettano dei messaggi contemporaneamente. Questo problema deve essere gestito come possibile dal codice dell'utente. Tuttavia le funzioni della classe in caso di conflitto impediscono la perdita di entrambi i messaggi (cosa che potrebbe portare a un blocco senza uscita se entrambi i programmi cercassero di reinviare subito il proprio messaggio). Dà quindi la priorità ai messaggi già arrivati a scapito di quelli in uscita, che potrebbero perdersi.

Lo schema sottostante mostara i momenti in cui non si può o non si dovrebbe trasmettere. Il primo schema si riferisce ai messaggi con richiesat di ACK, il secondo a quelli senza.

```
stato tx                    |*********|############ 1 ############|
A           ----------------|---------|---------------------------|---------->
|                         INVIA      ISR                           ISR
|                                    ISR              LEGGI    ISR
B           ----------------------------|-----------------|-------|---------->
stato tx              |####### 2 ########|                 |*******|
```

```
stato tx                    |*********|######## 3 #######|
A           ----------------|---------|------------------------------>
|                         INVIA      ISR
|                                    ISR              LEGGI
B           ----------------------------|-----------------|---------->
stato tx              |######## 2 #######|
```

: nessuna restrizione, è il momento giusto per trasmettere un messaggio

*** : impossibile trasmettere, invia() aspetta che sia di nuovo possibile (ma al massimo 50 ms)

: la funzione invia() non dovrebbe mai essere chiamata qui:

1. CHIAMATA AD `invia()` QUI –> PROBLEMA NEL CODICE DELL'UTENTE In teoria non bisognerebbe trasmettere (l'altra radio non è in modalità rx), ma in realtà se l'utente chiama invia() mentre la classe aspetta un ack per il messaggio precedente significa che l'utente ha rinunciato a controllare quell'ack. In tal caso invia() si comporta come se il messaggio precedente non avesse contenuto una richiesta di ack. Probabilmente questo messaggio andrà perso, ma il compito della funzione invia() non è aspettare l'ack precedente (quello è compito dell'utente, anche se lo aspettasse per un certo tempo invia() non potrebbe segnalare se è arrivato o no). La sequenza corretta sarebbe: invia() con richiesta ack -> aspettaAck(), che contiene un timeout -> ackRicevuto()? -> invia() prossimo messaggio, oppure invia() -> delay(x) -> ackRicevuto()? -> rinunciaAck() invia()
2. momento critico: se si chiama invia() qui ci sarà una collisione con l'invia() della radio A e entrambi i messaggi saranno persi, ma questa classe non ha modo di evitarlo. Spetta all'utente impedire queste collisioni o saperle gestire.
3. i messaggi inviati qui saranno persi. È un difetto dei messaggi senza ACK.

Hardware

Come già detto ho scritto questa classe in particolare per il modulo RFM69HCW di HopeRF, in commercio sia da solo sia inserito in altri moduli che offrono, ad esempio, un logic level shifting da 5V a 3.3V (ad es. Adafruit vende `https://www.adafruit.com/product/3071` per la maggior parte dei paesi, tra cui tutti quelli europei, e `https://www.adafruit.com/product/3070` per gli USA e pochi altri).

Il modulo comunica con il microcontrollore tramite SPI, deve poter chiamare un'interrupt su quest'ultimo e può "affidargli" il proprio pin di reset (non veramente sfruttato da questa classe, ma se è già connesso deve essere gestito per evitare reset indesiderati). Deve essere alimentato con una tensione di 3.3V.

RFM69	uC
MISO	MISO
MOSI	MOSI
SCK	SCK
NSS	I/O $*$
DIO0	INT $**$
RESET &	*I/O* $*$

- &: Opzionale
- $*$: OUT è qualsiasi pin di input/output (sarà configurato come output dalla classe)
- $**$: INT è un pin capace di attivare un interupt del microcontrollore. Ad esempio su Atmega328p, il microcontrollore di Arduino UNO, si possono usare i pin 4 e 5, cioé rispettivamente 2 e 3 nell'ambiete di programmazione Arduino.

Struttura messaggi

Tutti i messaggi inviati con le funzioni di questa classe hanno la seguente struttura:

Preamble	Sync word	Lunghezza	Intestazione	Contenuto	CRC
PREAMBLE_SIZE	SYNC_SIZE	1	1	lunghezza	2
01010101...	SYNC_VAL	lunghezza	intestazione	messaggio	crc

La prima riga è la lunghezzza della sezione in bytes, la seconda è il suo contenuto.

- `PREAMBLE_SIZE`, `SYNC_SIZE` e `SYNC_VAL` sono costanti definite nel file "RFM69_impostazioni.h".
- `lunghezza` e `messaggio` sono gli argomenti della funzione `invia()`.
- `intestazione` è un byte generato dalle funzioni di invio e letto da quelle di ricezione, inaccessibile all'utente.
- `crc` è un Cyclic Redundancy Checksum generato dalla radio.

Documentazione dettagliata dei membri della classe: RFM69 (pag. **13)**

Esempio di utilizzo

```
#include <Arduino.h>
#include "RFM69.h"


// #define MODULO_r o MODULO_t per compilare rispettivamente il programma per la
// radio ricevente o per quella trasmittente.
//--------------------------------------------------------------------------------
#define MODULO_r
```

```
// #define MODULO_t
//--------------------------------------------------------------------------------



// telecomando
#ifdef MODULO_r
// Pin SS, pin Interrupt, (eventualmente pin Reset)
RFM69 radio(2, 3);
// Un LED, 0 per non usarlo
#define LED 4
#endif

// quadricotetro
#ifdef MODULO_t
// Pin SS, pin Interrupt, (eventualmente pin Reset)
RFM69 radio(A2, 3, A3);
// Un LED, 0 per non usarlo
#define LED 7
#endif



void setup() {

    Serial.begin(115200);
    if(LED) pinMode(LED, OUTPUT);

    // Inizializza la radio. Deve essere chiamato una volta all'inizio del programma.
    // Restituisce 0
    int initFallita = radio.inizializza(4);
    if(initFallita) {
        // Stampa l'errore riscontrato (questa funzione pesa quasi 0.5 kB)
        radio.stampaErroreSerial(Serial, initFallita);
        // Inizializzazione fallita, blocca il progrmma
        while(true);
    }
}



#ifdef MODULO_t


void loop(){

    // crea un messaggio
    uint8_t lung = 4;
    uint8_t mess[lung] = {0,0x13, 0x05, 0x98};

    unsigned long t;
    bool ok;

    while(true) {

        // Aggiorna messaggio
        mess[0] = (uint8_t)radio.nrMessaggiInviati();

        Serial.print("Invio...");
        if(LED) digitalWrite(LED, HIGH);

        // Registra tempo di invio
        t = millis();

        // Invia
        radio.inviaConAck(mess, lung);
        // Aspetta fino alla ricezione di un ack o al timeout impostato nella classe
        while(radio.aspettaAck());
        // Controlla se è arrivato un Ack (l'attesa può finire anche senza ack, per timeout)
        if(radio.ricevutoAck()) ok = true;  else ok = false;

        // calcola il tempo trascorso dall'invio
        t = millis() - t;

        if(LED) digitalWrite(LED, LOW);

        if(ok) {
            Serial.print(" mess #");
            Serial.print(radio.nrMessaggiInviati());
            Serial.print(" trasmesso in ");
            Serial.print(t);
            Serial.print(" ms");
        }
        else {
```

```
            Serial.print(" messaggio #");
            Serial.print(radio.nrMessaggiInviati());
            Serial.print(" perso");
        }
        Serial.println();

        delay(1000);
    }
}

#endif


#ifdef MODULO_r


void loop(){

    // metti la radio in modalità ricezione
    radio.iniziaRicezione();

    // aspetta un messaggio
    while(!radio.nuovoMessaggio());

    if(LED) digitalWrite(LED, HIGH);

    // ottieni la dimensione del messaggio ricevuto
    uint8_t lung = radio.dimensioneMessaggio();
    // crea un'array in cui copiarlo
    uint8_t mess[lung];
    // leggi il messaggio
    int erroreLettura = radio.leggi(mess, lung);
    // ora `mess` contiene il messaggio e `lung` corrisponde alla lunghezza del
    // messaggio (in questo caso corrispondeve anche prima, ma avrebbe anche
    // potuto essere più grande, ad. es. se mess. fosse stato un buffer generico
    // già allocato alla dimensione del messaggio più lungo possibile)

    if (erroreLettura) {
        Serial.print("Errore lettura");
    }
    else {
        Serial.print("Messaggio (");
        Serial.print(lung);
        Serial.print(" bytes): ");
        for(int i = 0; i < lung; i++) {
            Serial.print(" 0x");
            Serial.print(mess[i], HEX);
        }
        Serial.print("  rssi: ");

        // Ottieni il valore RSSI del segnale che ha portato questo messaggio
        Serial.print(radio.rssi());
    }
    Serial.println();

    delay(50);
    if(LED) digitalWrite(LED, LOW);

}
#endif
```

Capitolo 2

Indice dei tipi composti

2.1 Elenco dei tipi composti

Queste sono le classi, le struct, le union e le interfacce con una loro breve descrizione:

Capitolo 3

Indice dei file

3.1 Elenco dei file

Questo è un elenco dei file documentati con una loro breve descrizione:

Capitolo 4

Documentazione delle classi

4.1 Riferimenti per la struct RFM69::Errore

Contenitore dell'`enum` **Errore::ListaErrori** (pag. 11).

`#include <RFM69.h>`

Tipi pubblici

- enum **ListaErrori** {
  ok = 0, **errore** = 1, **initTroppeRadio** = 2, **initInitSPIFallita** = 3,
  initNessunaRadioConnessa = 4, **initVersioneRadioNon0x24** = 5, **initPinInterruptNonValido** = 6, **initErroreImpostazione** = 7,
  inviaMessaggioVuoto = 8, **inviaTimeoutTxPrecedente** = 9, **leggiNessunMessaggio** = 10, **leggiArrayTroppoCorta** = 11,
  messaggioTroppoLungo = 12, **modImpossibile** = 13, **modTimeout** = 14 }

  Lista degli errori che le funzioni della classe possono restituire.

4.1.1 Descrizione dettagliata

Contenitore dell'`enum` **Errore::ListaErrori** (pag. 11).

vedi **ListaErrori** (pag. 11) per dettagli sui codici di errore

Definizione alla linea 759 del file RFM69.h.

4.1.2 Documentazione dei tipi enumerati (enum)

4.1.2.1 ListaErrori

`enum` **`RFM69::Errore::ListaErrori`**

Lista degli errori che le funzioni della classe possono restituire.

Tutte le funzioni di **RFM69** (pag. 13) che restituiscono un codice di errore utilizzano questa `enum` per definirlo. Si tratta delle funzioni seguenti:

- **`inizializza()`** (pag. 16)
- `inviaMessaggio()` (privata) e i suoi derivati pubblici (**`leggi()`** (pag. 20))
- **`leggi()`** (pag. 20)
- `cambiaModalita()` (privata) e i suoi derivati pubblici (**`iniziaRicezione()`** (pag. 20), **`standby()`** (pag. 23), ...)

I codici di errore vengono sempre forniti all'utente sotto forma di `int` per non costringerlo a usare questa enum se non vuole (i valori dei codici sono espliciti, dunque l'untente può limitarsi a stamparli e guardare a che errore corrispondono leggendo la definizione dell'`enum`).

Valori del tipo enumerato

ok	Nessun errore.
errore	Errore generico
initTroppeRadio	**inizializza()** (pag. 16): Visto che ha una sola ISR questa classe può gestrire una sola radio ma il programma ha provato ad inizializzarne una seconda
initInitSPIFallita	**inizializza()** (pag. 16): L'inizializzazione di SPI non è riuscita
initNessunaRadioConnessa	**inizializza()** (pag. 16): Non è stata trovato nessun dispositivo connesso a SPI con lo Slave Select specificato nel constructor
initVersioneRadioNon0x24	**inizializza()** (pag. 16): È stato trovato un dispositivo ma non è una radio RFM69
initPinInterruptNonValido	**inizializza()** (pag. 16): Il pin scelto come interrupt non va bene perché non è collegato ad alcun interrupt nel microcontrollore
initErroreImpostazione	**inizializza()** (pag. 16): Errore nella scrittura dei registri della radio
inviaMessaggioVuoto	inviaMessaggio(): La lunghezza del messaggio è nulla
inviaTimeoutTxPrecedente	inviaMessaggio(): **invia()** (pag. 17) è stata chiamata mentre c'era un messaggio in uscita, la funzione ha aspettato per più del tempo massimo di invio di un messaggio e alla fine dell'attesa il messaggio non era ancora partito
leggiNessunMessaggio	**leggi()** (pag. 20): Non c'è nessun nuovo messaggio da leggere
leggiArrayTroppoCorta	**leggi()** (pag. 20): l'array in cui la funzion **`leggi()`** (pag. 20) dovrerbbe copiare il messaggio è troppo corta per contenerlo
messaggioTroppoLungo	**leggi()** (pag. 20): il messaggio è troppo lungo per essere letto da questa istanza della classe, che è stata inizializzata con un buffer di dimensione inferiore
modImpossibile	cambiaModalita(): Nel contesto in cui è stata chiamata non è possibile passare alla modalità richiesta
modTimeout	cambiaModalita(): Il cambio di modalità non è avvenuto entro un tempo massimo ampiamente sufficiente.

Definizione alla linea 774 del file RFM69.h.

La documentazione per questa struct è stata generata a partire dal seguente file:

- RFM69/ **RFM69.h**

4.2 Riferimenti per la classe RFM69

Driver per i moduli radio RFM69.

`#include <RFM69.h>`

Composti

- struct **Errore**
  *Contenitore dell'`enum` **Errore::ListaErrori** (pag. 11).*

Membri pubblici

Constructor

- **RFM69** (uint8_t pinSS, uint8_t pinInterrupt)
  Constructor da usare se il pin RESET della radio non è connesso al uC.
- **RFM69** (uint8_t pinSS, uint8_t pinInterrupt, uint8_t pinReset)
  Constructor da usare se il pin RESET è connesso al uC.

Inizializzazione

Funzione di inizializzazione della classe, da chiamare prima di qualsiasi altra

- int **inizializza** (uint8_t lunghezzaMaxMessaggio)
  Inizializza la radio. Deve essere chiamato all'inizio del programma.

Funzioni fondamentali

Devono essere usate in ogni programma per permettere una comunicazione radio

- int **invia** (const uint8_t messaggio[ ], uint8_t lunghezza)
  Invia un messaggio.
- int **inviaConAck** (const uint8_t messaggio[ ], uint8_t lunghezza)
  Invia un messaggio con richiesta di ACK.
- int **inviaFinoAck** (const uint8_t messaggio[ ], uint8_t lunghezza, uint16_t &tentativi)
  Invia un messaggio ripetutamente fino alla ricezione di un ACK.
- int **leggi** (uint8_t messaggio[ ], uint8_t &lunghezza)
  Restituisce un messaggio, se ce n'è uno da leggere.
- int **iniziaRicezione** ()
  Attiva la radio in modo che possa ricevere dei messaggi.

Funzioni importanti

Probabilmente saranno chiamate in tutti i programmi, ma non sono strettamente indispensabili

- bool **nuovoMessaggio** ()
  Controlla se c'è un nuovo messaggio.
- uint8_t **dimensioneMessaggio** ()
  Restituisce la dimensione dell'ultimo mesasggio.
- bool **aspettaAck** ()
  Restituisce true se la classe sta aspettando un ACK.
- bool **ricevutoAck** ()
  Restituisce true se la radio ha ricevuto un ACK.
- void **rinunciaAck** ()
  Simula la ricezione di un ACK.

- int **standby** ()
  Mette la radio in standby (richiederà 1.25mA di corrente)

Funzioni ausiliarie

Utili ma non indispensabili

- int **listen** ()
  Mette la radio in modalità `listen`
- int **sleep** ()
  Mette la radio in modalità sleep (richiederà 0.1uA di corrente)
- void **sleepDefault** ()
  Imposta la modalità di default della radio.
- void **standbyDefault** ()
  Imposta la modalità di default della radio.
- void **listenDefault** ()
- void **rxDefault** ()
- void **impostaTimeoutAck** (uint16_t tempoMs)
  Imposta il tempo d'attesa massimo per un ACK.
- int8_t **rssi** ()
  Restituisce il valore RSSI per l'ultimo messaggio.
- uint8_t **tempoRicezione** ()
  Restituisce l'"ora" della ricezione dell'ultimo messaggio.

Log

Funzioni utili per monitorare il funzionamento della radio

- int **nrMessaggiInviati** ()
  Restituisce il numero di messaggi inviati dopo l'ultima inizializzazione.
- int **nrMessaggiRicevuti** ()
  Restituisce il numero di messaggi ricevuti dopo l'ultima inizializzazione.
- void **stampaErroreSerial** (HardwareSerial &serial, int errore)
  Stampa la descrizione di un errore sul monitor seriale.

Debug

Le seguenti funzioni permettono all'utente di leggere i registri della radio, normalmentenon sono utilizzate

- void **stampaRegistriSerial** (HardwareSerial &Serial)
  Stampa il valore di tutti i registri della radio sul monitor seriale.
- uint8_t **valoreRegistro** (uint8_t indirizzo)
  Leggi il valore di un registro della radio.

Destructor, copy constructor, copy operator

Queste funzioni servono per evitare perdite di memoria visto che la classe gestisce una risorsa allocata dinamicamente (il buffer) e non dovrebbe mai essere copiata

- ~**RFM69** ()
  Destructor.
- **RFM69** (const **RFM69** &)=delete
  Copy Constructor.
- **RFM69** & **operator=** (const **RFM69** &)=delete
  Copy Operator.

4.2.1 Descrizione dettagliata

Driver per i moduli radio RFM69.

Autore

Noè Archimede Pezzoli (noearchimede@gmail.com)

Data

Febbraio 2018

Per una descrizione approfondita, cfr. la pagina **Driver per i moduli radio RFM69** (pag. 1)

Definizione alla linea 404 del file RFM69.h.

4.2.2 Documentazione dei costruttori e dei distruttori

4.2.2.1 RFM69() [1/3]

```
RFM69::RFM69 (
            uint8_t pinSS,
            uint8_t pinInterrupt )
```

Constructor da usare se il pin RESET della radio non è connesso al uC.

Parametri

pinSS	Numero del pin Slave Select
pinInterrupt	Numero del pin attraverso il quale la radio genera un interrupt sul uC. Deve ovviamente essere un pin di interrupt.

Definizione alla linea 243 del file RFM69_inizializzazione.cpp.

4.2.2.2 RFM69() [2/3]

```
RFM69::RFM69 (
            uint8_t pinSS,
            uint8_t pinInterrupt,
            uint8_t pinReset )
```

Constructor da usare se il pin RESET è connesso al uC.

Parametri

pinSS	Numero del pin Slave Select
pinInterrupt	Numero del pin attraverso il quale la radio genera un interrupt sul uC. Deve ovviamente essere un pin di interrupt.
pinReset	Numero del pin collegato al pin RESET della radio.

Definizione alla linea 247 del file RFM69_inizializzazione.cpp.

4.2.2.3 ~RFM69()

```
RFM69::~RFM69 ( )
```

Destructor.

Dopo aver chiamato il destructor su un'istanza è possibile chiamare **inizializza()** (pag. 16) su un'altra senza ricevere l'errore `Errore::ListaErrori::initTroppeRadio`

Definizione alla linea 263 del file RFM69_inizializzazione.cpp.

4.2.2.4 RFM69() [3/3]

```
RFM69::RFM69 (
            const  RFM69 &  )  [delete]
```

Copy Constructor.

Deleted perché non ha senso copiare una radio

4.2.3 Documentazione delle funzioni membro

4.2.3.1 inizializza()

```
int RFM69::inizializza (
            uint8_t lunghezzaMaxMessaggio )
```

Inizializza la radio. Deve essere chiamato all'inizio del programma.

La funzione esegue le seguenti operazioni in questo ordine:

- controllo che la classe non debba gestire troppe radio (cioé più di una)
- prepara l'interfaccia SPI per comunicare con la radio
- (eventualmente esegue il reset della radio)
- controlla che un dispositivo sia connesso
- controlla che il dispositivo sia una radio RFM69
- collega l'interrupt della radio all'ISR della classe
- scrive tutti i registri della radio inserendovi le impostazioni stabilite nel file **RFM69_impostazioni.h** (pag. 30)
- inizializza alcune variabili della classe
- crea un buffer di `lunghezzaMaxMessaggio` bytes che resterà allocato fino alla distruzione dell'istanza della classe.

Nota

L'esecuzione di questa funzione richiede alcuni decimi di secondo.
Sarà allocata un'array di `lunghezzaMaxMessaggio` bytes.

Parametri

lunghezzaMaxMessaggio	Lunghezza massima dei messagi ricevuti da questa radio. Può essere diverso dalla lunghezza massima dei messaggi inviati (quindi da questo stesso parametro sull'altra radio)

Restituisce

Codice di errore definito nell'enum **RFM69::Errore::ListaErrori** (pag. 11)

Definizione alla linea 278 del file RFM69_inizializzazione.cpp.

4.2.3.2 invia()

```
int RFM69::invia (
            const uint8_t messaggio[],
            uint8_t lunghezza )
```

Invia un messaggio.

Questa funzione prepara il modulo radio per trasmettere un messaggio. La trasmissione inizierà alla fine di questa funzione e sarà terminata dall'isr().

Cfr. la descrizione generale della classe **RFM69** (pag. 13) per informazioni su quando questa funzione può o non può essere usata.

Parametri

messaggio[in]	array di bytes (`uint8_t`) che costituiscono il messaggio
lunghezza[in]	lunghezza del messaggio in bytes

Restituisce

Codice di errore definito nell'enum **RFM69::Errore::ListaErrori** (pag. 11)

Definizione alla linea 28 del file RFM69_comunicazione.cpp.

4.2.3.3 inviaConAck()

```
int RFM69::inviaConAck (
            const uint8_t messaggio[],
            uint8_t lunghezza )
```

Invia un messaggio con richiesta di ACK.

Il messaggio inviato conterrà una richiesta di ACK, che dovrà poi essere gestito da altre funzioni. Questa agisce come **invia()** (pag. 17), cioé prepara il modulo radio per trasmettere un messaggio. La trasmissione inizierà alla fine di questa funzione e sarà terminata dall'isr().

Cfr. la descrizione generale della classe **RFM69** (pag. 13) per informazioni su quando questa funzione può o non può essere usata.

Parametri

messaggio[in]	array di bytes (`uint8_t`) che costituiscono il messaggio
lunghezza[in]	lunghezza del messaggio in bytes

Restituisce

Codice di errore definito nell'enum **RFM69::Errore::ListaErrori** (pag. 11)

Definizione alla linea 36 del file RFM69_comunicazione.cpp.

4.2.3.4 inviaFinoAck()

```
int RFM69::inviaFinoAck (
            const uint8_t messaggio[],
            uint8_t lunghezza,
            uint16_t & tentativi )
```

Invia un messaggio ripetutamente fino alla ricezione di un ACK.

Questa funzione invia un messaggio, aspetta l'ACK e se non lo riceve entro il tesmpo specificato in **impostaTimeoutAck()** (pag. 24) lo invia di nuovo. Ripete questa operazione per al massimo `tentativi` volte, dopo le quali restituirà 1 (`Errore::ListaErrori::errore`) se il messaggio non è ancora stato confermato (quindi probabilmente non è arrivato). Restituirà invece 0 (`Errore::ListaErrori::ok`)subito dopo aver ricvevuto un ACK.

Dopo l'esecuzione della funzione `tentativi` conterrà il numero di tentativi effettuati.

Parametri

messaggio[in]	array di bytes (`uint8_t`) che costituiscono il messaggio
lunghezza[in]	lunghezza del messaggio in bytes
tentativi[in/out]	/b prima: numero di tentativi da effettuare prima di rinunciare alla trasmissione del messaggio /b dopo: numero di tentativi effettuati

Restituisce

Codice di errore definito nell'enum **RFM69::Errore::ListaErrori** (pag. 11)

Definizione alla linea 45 del file RFM69_comunicazione.cpp.

4.2.3.5 leggi()

```
int RFM69::leggi (
              uint8_t messaggio[],
              uint8_t & lunghezza )
```

Restituisce un messaggio, se ce n'è uno da leggere.

Il messaggio è trasferito dalla radio al microcontrollore già nell'isr(). Questa funzione restituisce all'utente il contenuto del buffer della classe senza accedere alla radio. Deve tuttavia utilizzarla se il messaggio ricevuto contiene una richiesta di ACK. In tal caso alla fine della funzione viene iniziata la trasmissione dell'ack. Se è già in corso una trasmissione (funzione **invia()** (pag. 17) chiamata prima di **leggi()** (pag. 20), normalmente non dovrebbe succedere), **leggi()** (pag. 20) aspetterà fino alla fine della trasmissione prima di inviare l'ack; dopo 50 ms interromperà ogni eventuale trasmissione.

Parametri

messaggio	array[out] in cui sarà copiato il messaggio. Deve essere almeno grande quanto il messaggio, del quale si può ottenere la lunghezza con la funzione **dimensioneMessaggio()** (pag. 21).
lunghezza	[in/out] lunghezza di messaggio. Dopo l'esecuzione della funzione corrisponderà esattamente alla dimensione del messaggio.

Restituisce

Codice di errore definito nell'enum **RFM69::Errore::ListaErrori** (pag. 11)

Definizione alla linea 133 del file RFM69_comunicazione.cpp.

4.2.3.6 iniziaRicezione()

```
int RFM69::iniziaRicezione ( )
```

Attiva la radio in modo che possa ricevere dei messaggi.

Per riceve la radio deve essere in modalità ricezione. Nelle altre modalità non si accorgerà nemmeno di aver ricevuto un messaggio. Quindi se non si chiama questa funzione sulla radio ricevente tutti i messaggi inviatile andranno persi.

Definizione alla linea 193 del file RFM69_comunicazione.cpp.

4.2.3.7 nuovoMessaggio()

```
bool RFM69::nuovoMessaggio ( )
```

Controlla se c'è un nuovo messaggio.

Restituisce

`true` se il buffer della classe contiene un nuovo messaggio.

Definizione alla linea 205 del file RFM69_comunicazione.cpp.

4.2.3.8 dimensioneMessaggio()

```
uint8_t RFM69::dimensioneMessaggio ( )
```

Restituisce la dimensione dell'ultimo mesasggio.

Restituisce

la dimensione dell'utlimo messaggio ricevuto in bytes

Definizione alla linea 213 del file RFM69_comunicazione.cpp.

4.2.3.9 aspettaAck()

```
bool RFM69::aspettaAck ( )
```

Restituisce true se la classe sta aspettando un ACK.

Restituisce

`true` se la classe sta aspettando un ack, cioé se ha inviato un messaggio con richiesta di ACK e non lo ha ancora ricevuto.

Questa funzione contiene un sistema di timeout. Dopo lo scadere del tempo restituisce `false` anche se non ha ricevuto nessun ack.

Avvertimento

`false` ha due signifcati opposti:

1. L'ACK è stato ricevuto
2. L'ACK non è arrivato, ma il tempo massimo di attesa ê scaduto. Per questo è necessario chiamare *sempore* anche **ricevutoAck()** (pag. 22).

Definizione alla linea 239 del file RFM69_comunicazione.cpp.

4.2.3.10 ricevutoAck()

```
bool RFM69::ricevutoAck ( )
```

Restituisce true se la radio ha ricevuto un ACK.

Restituisce

`true` se la radio ha ricevuto un ACK per l'ultimo messaggio inviato

Questa funzione non aspetta l'ACK, dice solo se è arrivato, quindi chiamata immediatamente dopo **`invia()`** (pag. 17) restituisce sempre `false`. Normalmente è perciò usata insieme a **aspettaAck()** (pag. 21) (cioé subito dopo di essa).

Esempio:

```
for(int i = 0; i < 3; i++) {
    invia();
    aspettaAck();
    if(ricevutoAck()) break;
}
if(!ricevutoAck()) {
    print("Trasmissione messaggio fallita");
}
```

Definizione alla linea 255 del file RFM69_comunicazione.cpp.

4.2.3.11 rinunciaAck()

```
void RFM69::rinunciaAck ( )
```

Simula la ricezione di un ACK.

Nota

Normalmente questza funzione non dovrebbe mai essere chiamata.

Questa funzione è chiamata automaticamente da **aspettaAck()** (pag. 21) allo scadere del tempo massimo. Può essere usata dall'utente per terminare l'attesa prima di quella scadenza. In caso di invio di un secondo messaggio prima della ricezione dell'ACK (ad es. per trasmettere un'informazione urgente) è chiamata automaticamente.

Se il vero ACK dovesse arrivare dopo l'esecuzione di questa funzione non avrà nessun effetto.

Definizione alla linea 262 del file RFM69_comunicazione.cpp.

4.2.3.12 standby()

```
int RFM69::standby ( )
```

Mette la radio in standby (richiederà 1.25mA di corrente)

La modalità `standby` serve per mettere la radio in pausa per qualche secondo. Per pause più lunghe conviene usare **`sleep()`** (pag. 23), soprattutto se l'intero dispositivo deve essere messo in standby per un certo periodo. Se invece il dispositivo utilizza relativamente tanta corrente durante lo standby della radio (ad es. ha accesi diversi LED, un motore, ...) la differenza tra **`sleep()`** (pag. 23) e **`standby()`** (pag. 23) non è rilevante.

Definizione alla linea 530 del file RFM69_comunicazione.cpp.

4.2.3.13 listen()

```
int RFM69::listen ( )
```

Mette la radio in modalità `listen`

`listen` è una modalità particolare che consiste in realtà nella continua alternanza tra due modalità: `rx` (ricezione) e `idle` (una specie di sleep adattato alla modalità listen).

Definizione alla linea 527 del file RFM69_comunicazione.cpp.

4.2.3.14 sleep()

```
int RFM69::sleep ( )
```

Mette la radio in modalità sleep (richiederà 0.1uA di corrente)

Definizione alla linea 533 del file RFM69_comunicazione.cpp.

4.2.3.15 sleepDefault()

```
void RFM69::sleepDefault ( )
```

Imposta la modalità di default della radio.

Questa funzione non cambia la modalità attuale!
Ad es.**sleepDefault()** (pag. 24); non corrisponde a **sleepDefault()** (pag. 24); **sleep()** (pag. 23);

Definizione alla linea 537 del file RFM69_comunicazione.cpp.

4.2.3.16 standbyDefault()

```
void RFM69::standbyDefault ( )
```

Imposta la modalità di default della radio.

Questa funzione non cambia la modalità attuale!
Ad es.**sleepDefault()** (pag. 24); non corrisponde a **sleepDefault()** (pag. 24); **sleep()** (pag. 23);

Definizione alla linea 540 del file RFM69_comunicazione.cpp.

4.2.3.17 listenDefault()

```
void RFM69::listenDefault ( )
```

Definizione alla linea 543 del file RFM69_comunicazione.cpp.

4.2.3.18 rxDefault()

```
void RFM69::rxDefault ( )
```

Definizione alla linea 546 del file RFM69_comunicazione.cpp.

4.2.3.19 impostaTimeoutAck()

```
void RFM69::impostaTimeoutAck (
            uint16_t tempoMs )
```

Imposta il tempo d'attesa massimo per un ACK.

Parametri

tempoMs	Tempo di attesa in millisecondi per la funzione **aspettaAck()** (pag. 21). Dopo aver atteso per questo tempo la funzione terminerà senza aver ricevuto un ACK.

Definizione alla linea 550 del file RFM69_comunicazione.cpp.

4.2.3.20 rssi()

```
int8_t RFM69::rssi ( )
```

Restituisce il valore RSSI per l'ultimo messaggio.

Restituisce

Received Signal Strength Indicator (RSSI) per l'ultimo messaggio ricevuto.

Definizione alla linea 219 del file RFM69_comunicazione.cpp.

4.2.3.21 tempoRicezione()

```
uint8_t RFM69::tempoRicezione ( )
```

Restituisce l'"ora" della ricezione dell'ultimo messaggio.

Restituisce

Il tempo in millisecondi dall'inizio del programma a cui è stata attivata l'`isr()` che segna la finem della ricezione del messaggio.

Definizione alla linea 224 del file RFM69_comunicazione.cpp.

4.2.3.22 nrMessaggiInviati()

```
int RFM69::nrMessaggiInviati ( )  [inline]
```

Restituisce il numero di messaggi inviati dopo l'ultima inizializzazione.

Restituisce

Il numero di messaggi inviati dopo l'ultima inizializzazione

Definizione alla linea 683 del file RFM69.h.

4.2.3.23 nrMessaggiRicevuti()

```
int RFM69::nrMessaggiRicevuti ( )  [inline]
```

Restituisce il numero di messaggi ricevuti dopo l'ultima inizializzazione.

Restituisce

Il numero di messaggi ricevuti dopo l'ultima inizializzazione

Definizione alla linea 687 del file RFM69.h.

4.2.3.24 stampaErroreSerial()

```
void RFM69::stampaErroreSerial (
            HardwareSerial & serial,
            int errore )
```

Stampa la descrizione di un errore sul monitor seriale.

Questa funzione permette di stampare sul monitor seriale la causa di un errore segnalato da una funzione di questa classe.

Funziona per i codici di errore definiti dall'`enum` **Errore::ListaErrori** (pag. 11).

Nota

Le stringhe di testo sono salvate nella memoria flash (del programma), non nella SRAM (delle variabili), e pesano circa 0.5kB.
Siccome contiene alcune centinaia di caratteri, questa funzione "pesa" più delle altre. Si consiglia perciò di usarla solo in fase di debug per poi liberare spazio quando la radio funziona.

Parametri

serial	un oggetto di `HardwareSerial`. Tipicamente sarà `Serial` (o ev. `Serial1` ecc. se si usa un Arduino Mega). errore codice di errore restituito da una delle funzioni sopra elencate.

Definizione alla linea 557 del file RFM69_comunicazione.cpp.

4.2.3.25 stampaRegistriSerial()

```
void RFM69::stampaRegistriSerial (
            HardwareSerial & Serial )
```

Stampa il valore di tutti i registri della radio sul monitor seriale.

Strampa il valore di tutti i registri della radio

Definizione alla linea 445 del file RFM69_inizializzazione.cpp.

4.2.3.26 valoreRegistro()

```
uint8_t RFM69::valoreRegistro (
            uint8_t indirizzo )
```

Leggi il valore di un registro della radio.

Definizione alla linea 473 del file RFM69_inizializzazione.cpp.

4.2.3.27 operator=()

```
RFM69& RFM69::operator= (
            const RFM69 & ) [delete]
```

Copy Operator.

Deleted perché non ha senso copiare una radio

La documentazione per questa classe è stata generata a partire dai seguenti file:

- RFM69/ **RFM69.h**
- RFM69/ **RFM69_comunicazione.cpp**
- RFM69/ **RFM69_inizializzazione.cpp**

Capitolo 5

Documentazione dei file

5.1 Riferimenti per il file RFM69/RFM69.h

Header della classe **RFM69** (pag. 13).

```
#include <Arduino.h>
#include <SPI.h>
```

Composti

- class **RFM69**

  Driver per i moduli radio RFM69.
- struct **RFM69::Errore**

  Contenitore dell'`enum` ***Errore::ListaErrori*** *(*pag. *11).*

5.1.1 Descrizione dettagliata

Header della classe **RFM69** (pag. 13).

5.2 Riferimenti per il file RFM69/RFM69_comunicazione.cpp

Implementazione delle principali funzioni pubbliche della classe **RFM69** (pag. 13).

```
#include "RFM69.h"
#include "RFM69_registri.h"
```

5.2.1 Descrizione dettagliata

Implementazione delle principali funzioni pubbliche della classe **RFM69** (pag. 13).

Questo file contiene l'implementazione della maggior parte delle funzioni pubbliche della classe. Ne sono escluse solo le funzioni di impostazione della radio, che utilizzano le costanti definite nel file **RFM69_impostazioni.↩ h** (pag. 30) che non è incluso in questo file (le funzioni qui non devono usare quelle costanti).

Il file è suddiviso in 5 sezioni:

1. Invio
2. Ricezione
3. ISR
4. ACK
5. Impostazioni

5.3 Riferimenti per il file RFM69/RFM69_costanti_impostazione.h

Costanti per l'impostazione della radio RFM69.

5.3.1 Descrizione dettagliata

Costanti per l'impostazione della radio RFM69.

Questo file contiene le scelte possibili per tutte le impostazioni dela radio **RFM69** (pag. 13) sotto forma di costanti o macro. Le costanti qui definite sono usate nel file **RFM69_impostazioni.h** (pag. 30), le macro nel file RFM69_↩ inizializzazioe.h

5.4 Riferimenti per il file RFM69/RFM69_impostazioni.h

File di configurazione del modulo radio RFM69.

5.4.1 Descrizione dettagliata

File di configurazione del modulo radio RFM69.

Questo file contiene una lunga serie di costanti `#define`d tramite le quali è possibile modificare tutte le impostazioni della radio. Il file è `#include`d in **RFM69_inizializzazione.cpp** (pag. 31), dove le sue costanti saranno usate per scrivere i registri della radio.

La maggior parte delle impostazioni sono liberamente selezionabili dall'utente.

5.5 Riferimenti per il file RFM69/RFM69_inizializzazione.cpp

Implementazione delle funzioni di inizializzazione della classe **RFM69** (pag. 13).

```
#include "RFM69.h"
#include "RFM69_registri.h"
#include "RFM69_impostazioni.h"
```

5.5.1 Descrizione dettagliata

Implementazione delle funzioni di inizializzazione della classe **RFM69** (pag. 13).

In questo file sono raggruppate tutte le funzioni di inizializzazione Il file è suddiviso in 5 sezioni:

1. Completamento delle impostazioni
2. Interpretazione dei dati del file di impostazione
3. Definizione dei membri static
4. Constructor e destructor
5. Inizializzazione
6. Debug

5.6 Riferimenti per il file RFM69/RFM69_registri.h

Registri della radio RFM69.

Definizioni

- #define **RFM69_00_FIFO** 0x00
- #define **RFM69_01_OP_MODE** 0x01
- #define **RFM69_02_DATA_MODUL** 0x02
- #define **RFM69_03_BITRATE_MSB** 0x03
- #define **RFM69_04_BITRATE_LSB** 0x04
- #define **RFM69_05_FDEV_MSB** 0x05
- #define **RFM69_06_FDEF_LSB** 0x06
- #define **RFM69_07_FRF_MSB** 0x07
- #define **RFM69_08_FRF_MID** 0x08
- #define **RFM69_09_FRF_LSB** 0x09
- #define **RFM69_0A_OSC_1** 0x0A
- #define **RFM69_0B_AFC_CTRL** 0x0B
- #define **RFM69_0D_LISTEN_1** 0x0D
- #define **RFM69_0E_LISTEN_2** 0x0E
- #define **RFM69_0F_LISTEN_3** 0x0F
- #define **RFM69_10_VERSION** 0x10
- #define **RFM69_11_PA_LEVEL** 0x11
- #define **RFM69_12_PA_RAMP** 0x12

- #define **RFM69_13_OCP** 0x13
- #define **RFM69_18_LNA** 0x18
- #define **RFM69_19_RX_BW** 0x19
- #define **RFM69_1A_AFC_BW** 0x1A
- #define **RFM69_1B_OOK_PEAK** 0x1B
- #define **RFM69_1C_OOK_AVG** 0x1C
- #define **RFM69_1D_OOK_FIX** 0x1D
- #define **RFM69_1E_AFC_FEI** 0x1E
- #define **RFM69_1F_AFC_MSB** 0x1F
- #define **RFM69_20_AFC_LSB** 0x20
- #define **RFM69_21_FEI_MSB** 0x21
- #define **RFM69_22_FEI_LSB** 0x22
- #define **RFM69_23_RSSI_CONFIG** 0x23
- #define **RFM69_24_RSSI_VALUE** 0x24
- #define **RFM69_25_DIO_MAPPING_1** 0x25
- #define **RFM69_26_DIO_MAPPING_2** 0x26
- #define **RFM69_27_IRQ_FLAGS_1** 0x27
- #define **RFM69_28_IRQ_FLAGS_2** 0x28
- #define **RFM69_29_RSSI_TRESH** 0x29
- #define **RFM69_2A_RX_TIMEOUT_1** 0x2A
- #define **RFM69_2B_RX_TIMEOUT_2** 0x2B
- #define **RFM69_2C_PREAMBLE_MSB** 0x2C
- #define **RFM69_2D_PREAMBLE_LSB** 0x2D
- #define **RFM69_2E_SYNC_CONFIG** 0x2E
- #define **RFM69_2F_SYNC_VALUE_1** 0x2F
- #define **RFM69_30_SYNC_VALUE_2** 0x30
- #define **RFM69_31_SYNC_VALUE_3** 0x31
- #define **RFM69_32_SYNC_VALUE_4** 0x32
- #define **RFM69_33_SYNC_VALUE_5** 0x33
- #define **RFM69_34_SYNC_VALUE_6** 0x34
- #define **RFM69_35_SYNC_VALUE_7** 0x35
- #define **RFM69_36_SYNC_VALUE_8** 0x36
- #define **RFM69_37_PACKET_CONFIG_1** 0x37
- #define **RFM69_38_PAYLOAD_LENGHT** 0x38
- #define **RFM69_39_NODE_ADRS** 0x39
- #define **RFM69_3A_BROADCAST_ADRS** 0x3A
- #define **RFM69_3B_AUTO_MODES** 0x3B
- #define **RFM69_3C_FIFO_TRESH** 0x3C
- #define **RFM69_3D_PACKET_CONFIG_2** 0x3D
- #define **RFM69_3E_AES_KEY_1** 0x3E
- #define **RFM69_3F_AES_KEY_2** 0x3F
- #define **RFM69_40_AES_KEY_3** 0x40
- #define **RFM69_41_AES_KEY_4** 0x41
- #define **RFM69_42_AES_KEY_5** 0x42
- #define **RFM69_43_AES_KEY_6** 0x43
- #define **RFM69_44_AES_KEY_7** 0x44
- #define **RFM69_45_AES_KEY_8** 0x45
- #define **RFM69_46_AES_KEY_9** 0x46
- #define **RFM69_47_AES_KEY_10** 0x47
- #define **RFM69_48_AES_KEY_11** 0x48
- #define **RFM69_49_AES_KEY_12** 0x49
- #define **RFM69_4A_AES_KEY_13** 0x4A
- #define **RFM69_4B_AES_KEY_14** 0x4B
- #define **RFM69_4C_AES_KEY_15** 0x4C
- #define **RFM69_4D_AES_KEY_16** 0x4D

- #define **RFM69_4E_TEMP_1** 0x4E
- #define **RFM69_4F_TEMP_2** 0x4F
- #define **RFM69_58_TEST_LNA** 0x58
- #define **RFM69_5A_TEST_PA_1** 0x5A
- #define **RFM69_5C_TEST_PA_2** 0x5C
- #define **RFM69_6F_TEST_DAGC** 0x6F
- #define **RFM69_71_TEST_AFC** 0x71
- #define **RFM69_RESERVED**(x)
- #define **RFM69_ULTIMO_REGISTRO** 0x71
- #define **RFM69_FLAGS_1_MODE_READY** 0x80
- #define **RFM69_FLAGS_1_RX_READY** 0x40
- #define **RFM69_FLAGS_1_TX_READY** 0X20
- #define **RFM69_FLAGS_1_PLL_LOCK** 0X10
- #define **RFM69_FLAGS_1_RSSI** 0X08
- #define **RFM69_FLAGS_1_TIMEOUT** 0X04
- #define **RFM69_FLAGS_1_AUTO_MODE** 0X02
- #define **RFM69_FLAGS_1_SYNC_ADDR_MATCH** 0X01
- #define **RFM69_FLAGS_2_FIFO_FULL** 0x80
- #define **RFM69_FLAGS_2_FIFO_NOT_EMPTY** 0x40
- #define **RFM69_FLAGS_2_FIFO_LEVEL** 0x20
- #define **RFM69_FLAGS_2_FIFO_OVERRUN** 0x10
- #define **RFM69_FLAGS_2_PACKET_SENT** 0x08
- #define **RFM69_FLAGS_2_PAYLOAD_READY** 0x04
- #define **RFM69_FLAGS_2_CRC_OK** 0x02

5.6.1 Descrizione dettagliata

Registri della radio RFM69.

Questo file contiene l'elenco `#define`d dei registri della radio RFM69.

5.6.2 Documentazione delle definizioni

5.6.2.1 RFM69_RESERVED

```
#define RFM69_RESERVED(
              x )
```

Valore:

```
( \
    x == 0x0C || \
    (0x13 < x && x < 0x18) || \
    (0x4F < x && x < 0x58) || \
    x == 0x59 || \
    x == 0x5B || \
    (0x5c < x && x < 0x6F) || \
    (0x6F < x && x < 0x71) || \
0x71 < x )
```

Definizione alla linea 103 del file RFM69_registri.h.

5.7 Riferimenti per il file RFM69/RFM69_SPI.cpp

Implementazione della comunicazione SPI con la radio RFM69.

```
#include "RFM69.h"
```

5.7.1 Descrizione dettagliata

Implementazione della comunicazione SPI con la radio RFM69.

Il file contiene l'implementazione della classe che gestisce la comunicazione tra la radio RFM69 e il microcontrollore. Questa classe è un membro privato della classe **RFM69** (pag. 13).

Indice analitico